

语法分析器-实验报告

2021201709 李俊霖

在上一个词法实验的基础上，借助yacc工具实现一个语法分析器。

1.实验过程

1.1.修改词法分析（morphology.l）

- 由于在之前的词法分析实验中，只是将关键字、标识符、算符等归为一个大类，但是在语法分析过程中，需要将关键字和算符单独分出一符一种，一遍与进行LR语法分析。
- 因此，本次实验的第一步便是单独将关键字和算符单独分出，并存入语法表格中，返回相对应的关键字（全局变量）。
- 示例如下（由于代码较长，故在此仅展示部分改动）

```
IF "if"
ELIF "else if"
ELSE "else"
WHILE "while"
DO "do"

%%
{IF} {col+=yyleng; yylval=++cnt;strcpy(L[cnt], yytext);return IF;}
{ELIF} {col+=yyleng; yylval=++cnt;strcpy(L[cnt], yytext);return ELIF;}
{ELSE} {col+=yyleng; yylval=++cnt;strcpy(L[cnt], yytext);return ELSE;}
{WHILE} {col+=yyleng; yylval=++cnt;strcpy(L[cnt], yytext);return WHILE;}
{DO} {col+=yyleng; yylval=++cnt;strcpy(L[cnt], yytext);return DO;}
```

1.2.完成语法分析（grammar.y）

yacc 采用 LR（1）（实际上是 LALR（1））语法分析方法。

- 四则运算优先级设定：

定义多组 %left 或 %right，在后面定义的组有更高的优先级。

```
%left '+' '-'
%left '*' '/'
```

此操作可实现乘除的优先级比加减要高。

- If-Else 冲突

当有两个IF一个ELSE时，让ELSE与最近的IF匹配。

示例：给IF-ELSE语句比IF语句更高的优先级：

```
%nonassoc IFX
%nonassoc ELSE
stmt: IF expr stmt %prec IFX
    | IF expr stmt ELSE stmt
```

- 按照文档编写语法规则和对应的值传递。
- 程序输出将归约项的编号（终结符/非终结符）的值输出，便于后续绘制抽象语法树构造连边关系。
 - 一个语法规则的示例如下：

```
CompUnit:
    otherCompUnit {
        $$ = ++cnt;
        strcpy(L[$$], "CompUnit");
        printf ("%d:CompUnit -> %d:OtherCompUnit\n", $$, $1);
    }
    | CompUnit OtherCompUnit {
        $$ = ++cnt;
        strcpy(L[$$], "CompUnit");
        printf ("%d:CompUnit -> %d:CompUnit %d:OtherCompUnit\n", $$, $1, $2);
    }
;
```

1.3.生成语法分析 .txt 文件

使用脚本 `run.sh` 生成输出文件。

```
# 此脚本用于词法、语法分析，生成out.txt文件
flex ./morphology.l
yacc -d ./grammar.y
gcc y.tab.c lex.yy.c -o mc -O2 -w
./mc ./test/3.sy > ./test/3-out.txt
```

输出结果示例如下：（部分）

```
6:PrimaryExp -> 5:10
7:UnaryExp -> 6:PrimaryExp
8:MulExp -> 7:UnaryExp
10:AddExp -> 8:MulExp
11:ConstArrayIndex -> 4:\= 10:AddExp
12:VarDecl -> 1:const 2:int 3:pad 11:ConstArrayIndex
16:PrimaryExp -> 15:20
17:UnaryExp -> 16:PrimaryExp
18:MulExp -> 17:UnaryExp
20:AddExp -> 18:MulExp
21:ConstArrayIndex -> 14:\= 20:AddExp
```

1.4.绘制抽象语法树

1.4.1. 生成 .dot 文件

- 使用 `python` 编写程序，将语法分析器的输出 `.txt` 文件转换为 `graphviz` 工具可识别的 `.dot` 文件。
- `python` 代码见 `txt2dot.ipynb` 文件。
- 主要思路为：

- 先将语法分析器的输出结果反序（因为语法分析器的生成顺序为归约顺序，因此是自底向上的，但绘制语法树时要先将出边节点定义，因此是自顶向下的）
- 解析语法分析器的输出结果，按照节点和连边关系生成 `.dot` 文件。
- 1.dot 文件示例如下（部分）

```
digraph " "{
node [shape = record,height=.1]
node0[label = "<f0> CompUnit"];
node1[label = "<f0> CompUnit|<f1> OtherCompUnit"];
"node0":f0->"node1";
node2[label = "<f0> FuncDef"];
"node1":f1->"node2";
node3[label = "<f0> int|<f1> main|<f2> \(|<f3> \)|<f4> Block"];
"node2":f0->"node3";
node4[label = "<f0> \{|<f1> BlockItem|<f2> \}"];
"node3":f4->"node4";
node5[label = "<f0> Stmt|<f1> BlockItem"];
"node4":f1->"node5";
}
```

1.4.2.绘制图像

使用指令 `dot -Tpng -o 1.png 1.dot` 将 1.dot 文件转化为 1.png 文件。

图像如下：（由于图像较大，故不在此一一展示，详情请见压缩包）

- 样例2：



- 样例3:



1. 通过本次语法实验，我对语法分析部分的知识有了更深入的理解，特别是LR语法分析的语法规则，也学会使用yacc工具辅助语法分析器的实现。
2. 在处理表达式和IF/ELSE语句冲突的过程中，我也对归约-归约冲突、移进-归约冲突有了更深理解，对语法分析的过程更熟悉。